Computational aspects of assigning agents to a line

Aziz, Haris; Hougaard, Jens Leth; Moreno-Ternero, Juan D.; Østerdal, Lars Peter Raahave

Published in:
Mathematical Social Sciences

DOI:
10.1016/j.mathsocsci.2017.02.004

Publication date:
2017

Document version
Publisher's PDF, also known as Version of record

Mathematical Social Sciences 90 (2017) 93–99

Contents lists available at ScienceDirect

Mathematical Social Sciences

journal homepage: www.elsevier.com/locate/mss

Computational aspects of assigning agents to a line☆

Haris Aziz [a,b], Jens Leth Hougaard [c], Juan D. Moreno-Ternero [d,e], Lars Peter Østerdal [f,*]

[a] Data61, Australia
[b] University of New South Wales, Australia
[c] University of Copenhagen, Denmark
[d] Universidad Pablo de Olavide, Spain
[e] CORE, Université catholique de Louvain, Belgium
[f] Copenhagen Business School, Denmark

HIGHLIGHTS

- We consider the problem of assigning agents to slots on a line.
- We introduce an approach to compute aggregate gap-minimizing assignments.
- We also extend the approach to gap-egalitarian and probabilistic assignments.
- The approach relies on an algorithm which is faster than general purpose algorithms.

ARTICLE INFO

Article history:
Available online 17 February 2017

ABSTRACT

We consider the problem of assigning agents to slots on a line, where only one agent can be served at a slot and each agent prefers to be served as close as possible to his target. We introduce a general approach to compute aggregate gap-minimizing assignments, as well as gap-egalitarian assignments. The approach relies on an algorithm which is shown to be faster than general purpose algorithms for the assignment problem. We also extend the approach to probabilistic assignments and explore the computational features of existing, as well as new, methods for this setting.

1. Introduction

An assignment problem refers to the broad situation in which a set of objects are to be allocated among a group of agents, and each agent is to receive exactly one object. The interest on problems of this sort, which abound in real life, ranges from ancient writings to modern scientific research in different fields (Hylland and Zeckhauser, 1979; Hofstee, 1990; Bogomolnaia and Moulin, 2001; Burkhard et al., 2009).

In a recent paper, Hougaard et al. (2014) introduce and analyze a specific assignment problem, known as the problem of *assigning agents to a line*. In such a problem, each agent has a preferred slot (target) and wants to be served as close as possible to it. Each slot can serve only one agent. The number of slots is at least the number of agents but can be arbitrarily larger than the number of agents.

Hougaard et al. (2014) focus on the notion of (aggregate) *gap minimization* for such a problem, i.e., minimizing the sum of the distances. More precisely, they provide a direct method for testing if a given deterministic assignment is aggregate gap-minimizing, and make use of it to propose an aggregate gap-minimizing modification of the classic *random priority* method to solve this class of problems. It is shown that aggregate gap minimization is incompatible with sd-no-envy and sd-strategy-proofness. Moreover, it is shown that the results extend to more general preference structures. However, the computational aspects of such methods are yet to be explored, and we aim to do so in this paper. Nevertheless, the aims of this paper, which we enumerate next, will go beyond

DOI of original article: http://dx.doi.org/10.1016/j.mathsocsci.2016.12.004.

☆ Paper was prepared for Herve Moulin's Festschrift. We thank him for being a constant source of inspiration for our research. We also thank Maurice Queyranne and two anonymous referees for helpful comments and suggestions on this paper. Financial support from the Danish Council for Independent Research (Grant ID: DFF-1327-0009) and the Spanish Ministry of Economy and Competitiveness (ECO2014-57413-P) is gratefully acknowledged.

* Corresponding author.
E-mail address: lpo.eco@cbs.dk (L.P. Østerdal).

http://dx.doi.org/10.1016/j.mathsocsci.2017.02.004
0165-4896/

studying the computational aspects of the methods developed in Hougaard et al. (2014).

First, we are concerned with the more general problem of identifying (rather than just testing) aggregate gap-minimizing assignments. We introduce an algorithm, dubbed as the *Neat Shifting Algorithm*, to compute aggregate gap-minimizing assignments, which is shown to be faster than general purpose algorithms for the assignment problem. The algorithm relies on the concept of *neatness*, which refers to target-ordered assignments in which all agents with the same target are placed next to each other.

Second, we are also concerned with identifying *gap-egalitarian* assignments, i.e., assignments lexicographically minimizing the distribution of gaps.[1] We show that gap-egalitarian and aggregate gap-minimizing assignments may not coincide; in particular, a gap-egalitarian assignment may not be aggregate gap-minimizing and an aggregate gap-minimizing assignment may not be gap-egalitarian. We also show that, whereas neatness is a necessary condition for gap-egalitarian assignments, there always exists at least one aggregate gap-minimizing assignment that is neat. Furthermore, we show that a suitable adaptation of the *Neat Shifting Algorithm* can be used to yield gap-egalitarian assignments.

Third, when focusing on probabilistic assignments, which arise when randomizing among outcomes, we present new methods, each having their own merits.

The rest of the paper is organized as follows. In Section 2, we discuss some related literature to this paper. In Section 3, we introduce the model and preliminary concepts. In Section 4, we focus on aggregate gap-minimizing assignments and introduce the Neat Shifting Algorithm. In Section 5, we extend the analysis to probabilistic assignments and present aggregate gap-minimizing modifications of the canonical random priority solution. In Section 6, we focus on gap-egalitarian assignments. Section 7 concludes the paper.

2. Related work

As mentioned above, the closest work to this paper is Hougaard et al. (2014) in which the problem of assigning agents to a line is considered. Therein, a direct method for testing if a given deterministic assignment is aggregate gap-minimizing is provided. Chun and Park (forthcoming) consider the same model but assume cardinal (and not just ordinal) preferences. More precisely, they assume that each agent's utility is equal to the amount of monetary transfer minus the distance from the target to his assigned slot. They look at aggregate gap-minimizing as well as gap-egalitarian assignments. They obtain some characterizations but, in contrast to Hougaard et al. (2014), they assume that the number of slots is equal to the number of agents, which greatly simplifies the structure of the assignments as well as the complexity of the problem.[2]

In related work, Procaccia and Tennenholtz (2013) consider a similar problem in the context of 'facility location' and pursue the goals of minimizing the aggregate gap, as well as minimizing the maximum gap. Instead of considering optimal solutions, the focus is on *strategy-proof* mechanisms that provide good approximations of the optimal solutions.

The problem of assigning agents to a line is a slightly different version of the model introduced by Bogomolnaia and Moulin (2001), which could be considered as one of the most influential papers on assignment problems within the economic literature. We not only allow for weak preferences, as Katta and Sethuraman (2006), but we actually assume that preferences are *single-peaked*, as Kasajima (2013) does, and symmetric (to both sides of the peak).

Assignment problems have long been analyzed within the economics literature, mostly focusing on issues of efficiency, incentive compatibility and fairness. Hylland and Zeckhauser (1979) proposed an algorithm, based on market-clearing prices, for allocating individuals to positions with limited capacities. The algorithm guarantees (ex-ante) efficiency, and envy-freeness, but fails to be strategy-proof. As a matter of fact, Zhou (1990) showed that no solution in such a setting satisfies strategy-proofness, (ex-ante) efficiency, and a notion of fairness weaker than envy-freeness. Bogomolnaia and Moulin (2001) restricted attention to strict preferences and ordinal solutions and introduced the notion of *sd-efficiency*, which states that a probabilistic assignment is not stochastically dominated with respect to individual preferences over certain objects. They characterized all sd-efficient assignments and showed that sd-efficiency is incompatible with sd-strategy-proofness, and equal treatment of equals.[3] They also showed that, for more than three agents, no solution satisfies sd-envy-freeness and sd-strategy-proofness together, along with equal treatment of equals.

Previous papers in the economics literature have largely ignored computational aspects. The literature on operations research, computer science and artificial intelligence, on the other hand, has covered various computational aspects of assignment problems (Burkhard et al., 2009; Bouveret et al., 2016; Nguyen et al., 2014; Lang and Rothe, 2015). When the number of items to be allocated to the agents is equal to the number of agents, standard maximum weight matching algorithms can be used to compute allocations that maximize total welfare. For maximizing egalitarian welfare, a perfect matching algorithm can be used to compute a maximum egalitarian assignment. However, the problem of maximizing egalitarian welfare is NP-complete if the number of items is more than the number of agents and agents can get multiple items (Demko and Hill, 1988). Aziz et al. (2016) proved that testing Pareto optimality is coNP-complete under additive preferences even if each agent is to be allocated two items. Aziz et al. (2013) presented a general algorithm to compute and test Pareto optimal outcomes in discrete allocation and coalition formation settings with ordinal preferences. More recently, Damamme et al. (2015) investigated the power of dynamics based on rational bilateral deals (swaps) in various settings including ours. Among other things, they prove NP-hardness of deciding whether a utilitarian or egalitarian allocation is reachable. Now, as they acknowledge, our model of assigning agents to slots on a line induces, by default, a more restrictive domain, as the notion of distance is symmetric.

3. Preliminaries

We consider the model introduced by Hougaard et al. (2014). Imagine a facility with a fixed service capacity that can serve one agent at each (equidistant) *slot*. Agents are labeled by letters, $A, B, \ldots$ with generic elements i and j. The number of slots is infinite and identified with the integer numbers. Each agent i has a preferred slot t_i which we refer to as the agent's *target*. We label the agents so that $t_A \leq t_B \leq \cdots$. A *problem of assigning agents to a line* (in short, a *problem*), consists of a set of agents and the list of their corresponding targets (i.e., slots at which they would like to be served).

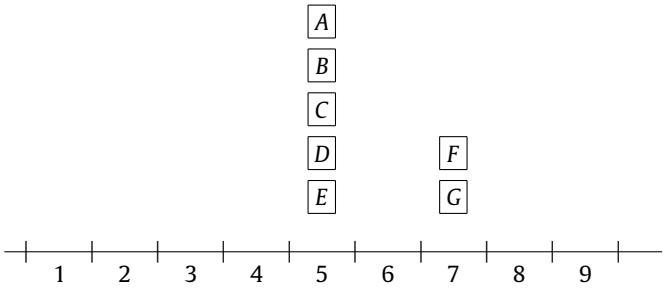
[1] More precisely, we consider first assignments in which the largest gap is as small as possible. Among those, we consider the assignments in which the second largest gap is as small as possible, etc.

[2] The computational aspects of the methods introduced in those papers are not considered therein.

[3] Abdulkadiroğlu and Sönmez (2003) provided an alternative characterization of sd-efficiency.

In general, a problem can be described, and depicted graphically, as in Example 1. Note that the input to the problem is linear in the number of agents.

Example 1. $[\{A, B, C, D, E\} : 5; \{F, G\} : 7]$. Five agents wish to be served at slot 5, and two agents wish to be served at slot 7.



An *allocation* (of a problem) is a deterministic assignment of agents to slots. Denote by x_i the slot assigned to agent i in allocation x. We shall often refer to x_i as agent i's outcome (in allocation x). For any allocation x and any two distinct agents A, B, it must be the case that $x_A \neq x_B$. For each allocation x we define the *gap* of agent i as $g_i(x) = |t_i - x_i|$.

An allocation is (aggregate) *gap-minimizing* if it minimizes the aggregate gap of the agents.

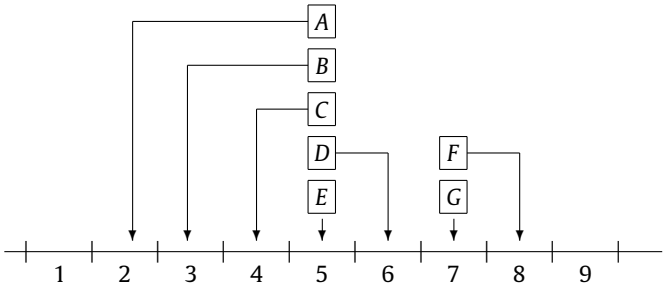
For each assignment P, we denote by $v(P)$ the vector of individual gaps in decreasing order. We refer to $v(P)$ as the *signature vector* of P. We say that an assignment is *gap-egalitarian* if it has the lexicographically minimum signature vector. Formally, given $\alpha, \beta \in \mathbb{R}^n$ satisfying $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$, $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$, we say that α *lexicographically dominates* β if there exists $k \in \{1, 2, \ldots, n\}$ such that $\alpha_i = \beta_i$, for each $i = 1, \ldots, k-1$, and $\alpha_k < \beta_k$. In words, the $k-1$ largest coordinates of α and β coincide and the kth largest coordinate of α is smaller than that of β.[4] Then, an assignment P is *gap-egalitarian* if its signature vector $v(P)$ is not *lexicographically dominated* by the signature vector $v(Q)$, of any other assignment Q.

The next figures provide instances of aggregate gap-minimizing and gap-egalitarian allocations for the problem in Example 1.

Example 1 **(cont'd)**:

(a) Allocation x is aggregate gap-minimizing :

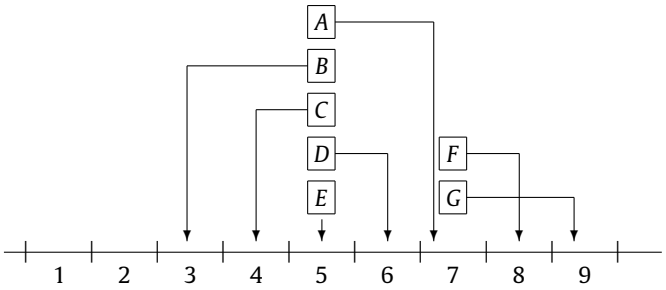
$x \in \arg\min \sum_i g_i(z)$



(b) Allocation y is gap-egalitarian :

$y \in \arg\ \text{lexmin}\{g_i(z) : i = A, B, C, \ldots\}$

[4] For $k = 1$, we simply say that the largest coordinate of α is smaller than the largest coordinate of β.



4. Aggregate gap-minimizing assignments

Hougaard et al. (2014) provided an algorithm to test whether an assignment is (aggregate) gap-minimizing. Building onto it, we now propose a new algorithm to actually determine (aggregate) gap-minimizing assignments. The precise definition of the algorithm, which we call the *Neat Shifting Algorithm*, is introduced below.

We first present some notation and preliminary results that are necessary to understand the definition, as well as the main features of the algorithm.

For each problem, agents can be partitioned into *types* according to their targets. Formally, for each target k, let N_k denote the set of agents sharing such a target k. Then, $N = \bigcup_k N_k$. Each assignment in which agents from the same type N_k are placed contiguously can be represented compactly by stating the leftmost slot $\ell(N_k)$ and the rightmost slot $r(N_k)$ of the type. Each assignment in which all agents of the same type are located contiguously can be represented in the *type form*. We say that an assignment is *target-ordered* if it is not the case that there exist two agents i and j where i is located to the left of j but t_i is to the right of t_j. We say that an assignment is *contiguous* if all agents with the same target are placed next to each other. Finally, we say that an assignment is *neat* if it is both target-ordered and contiguous. Then, we have the following:

Lemma 1. *There always exists a target-ordered aggregate gap minimizing assignment. Moreover, such an assignment is contiguous. Thus, there always exists a neat aggregate gap-minimizing assignment.*

Proof. Assume there is an aggregate gap minimizing assignment in which an agent i is located to the left of another agent j, but t_i is to the right of t_j. If so, swapping the agents cannot increase the aggregate gap. Now, assume there is a target-ordered aggregate gap minimizing assignment that is not contiguous. This means that there exist agents of at least one type that have empty slots between them. Now, the target of these agents can either be within the gap, to the left of the gap or to right of the gap. In all three cases, the aggregate gap can be decreased by moving agents one slot towards the gap. ■

The *Neat Shifting Algorithm* yields aggregate gap-minimizing assignments by considering neat assignments. The basic idea of the algorithm is that types are first to be placed exactly at, and then around, the target. In case these slots are already taken, the subset of agents is placed as close to the target as possible in the available slots to the right. The new type is pushed towards the target, which leads to agents in the way being shifted as well. If such a shift does not increase the aggregate gap, then it is implemented and shifts are repeated.

The next lemma shows that assignments represented in type form can be computed in linear time on the number of agents.

Lemma 2. *For any assignment represented in its type form, its gap can be computed in time $O(n)$.*

Algorithm 1 Neat Shifting Algorithm

Input: Agent set $N = \{1, \ldots, n\}$ with target t_i for each $i \in N$, and goal G which is the optimality criteria that is either aggregate gap-minimization or egalitarian gap-minimization.
Output: Allocation.

Partition N into sets $N_1, \ldots, N_{k'}$ where the target of agents in N_i is t_i and $t_i < t_j$ for $i < j$. In the solution computed, all agents in N_i will be to the left of all agents in N_j for $i < j$.
Initialize $k = 0$.
while $k < k'$ **do**
  Set $\ell(N_{k+1})$ to $\max(r(N_k) + 1, t_{k+1} - \lfloor n_{k+1}/2 \rfloor)$ and set $r(N_{k+1})$ to $\ell(N_{k+1}) + n_{k+1} - 1$.
  if $\ell(N_{k+1}) = t_{k+1} - \lfloor n_{k+1}/2 \rfloor$ **then**
    N_{k+1} was placed in the optimal position without compromising previous sets of agents.
  else if $\ell(N_{k+1}) > t_{k+1} - \lfloor n_{k+1}/2 \rfloor$ **then**
    Set leftplace to true
    while leftplace **do**
      Set ℓ' to ℓ and r' to r values.
      set $r'(N_{k+1})$ to $r'(N_{k+1}) - 1$ and $\ell'(N_{k+1})$ to $\ell'(N_{k+1}) - 1$.
      Set j' to $k + 1$.
      while $(\ell'(N_{j'}) = r(N_{j'-1}))$ **do**
        set $r'(N_{j'-1})$ to $r'(N_{j'-1}) - 1$ and $\ell'(N_{j'-1})$ to $\ell'(N_{j'-1}) - 1$.
        decrease j' by one.
      end while
      if G wrt ℓ' and r' values is same as wrt ℓ and r values **then**
        Set ℓ and r values to ℓ' and r' values
      else
        Set leftplace to false.
      end if
    end while
    Increase k
  end if
end while

Proof. We initialize the gap G to zero. We can simply go through all the agents and for each agent, we add the difference between the agent's position and the agent's target slot number to G. ■

We say that a neat aggregate gap-minimizing assignment is *left-respecting* if there does not exist a different neat aggregate gap-minimizing assignment for which all types' segments are moved to the left, or stay fixed.[5]

Lemma 3. *There exists a unique (subject to intra-type swapping) left-respecting aggregate gap-minimizing assignment.*

Proof. For a given problem, let $\mathcal{A}$ denote the set of its neat aggregate gap minimizing assignments. For each assignment x, let $\ell(N_k, x)$ be the leftmost slot occupied by an agent in type N_k. We claim that there exists $y \in \mathcal{A}$ such that, for each N_k, $\ell(N_k, y) = \min_{x \in \mathcal{A}} \ell(N_k, x)$. Let $i = 1, \ldots, k'$ be an index on types where 1 is the type with target furthest to the left and k' is the type with target furthest to the right. We proceed by induction on types.

Type 1. Obviously, there exists an assignment $y \in \mathcal{A}$ where $\ell(N_1, y) = \min_{x \in \mathcal{A}} \ell(N_1, x)$.

Induction hypothesis. Assume there exists an assignment $y \in \mathcal{A}$ where $\ell(N_k, y) = \min_{x \in \mathcal{A}} \ell(N_k, x)$, for $i = 1, \ldots, k$.

Type $k + 1$. Assume, by contradiction, that $\ell(N_{k+1}, y) \neq \min_{x \in \mathcal{A}} \ell(N_{k+1}, x)$. Then, there exists an assignment $z \in \mathcal{A}$ where $\ell(N_{k+1}, z) = \min_{x \in \mathcal{A}} \ell(N_{k+1}, x)$. Now, define the assignment w such that it is identical to y for types $1, \ldots, k$ and identical to z for types $k+1, \ldots, k'$. We claim that $w \in \mathcal{A}$. Assume, by contradiction, that this is not the case. Then, going from w to y would reduce the aggregate gap, hence contradicting that $z \in \mathcal{A}$. ■

Theorem 1. *The Neat Shifting Algorithm yields aggregate gap-minimizing assignments.*

Proof. We need to prove that the assignment returned by the algorithm minimizes the aggregate gap. First of all, note that the assignment produced by the algorithm is neat.

In the first iteration, the algorithm produces a left-respecting neat gap-minimizing assignment for N_1. Suppose this is the case for iterations $1, \ldots, k$. Now, consider the result of iteration $k + 1$, which we aim to show is left-respecting and aggregate gap-minimizing. Indeed, we first try to place the block of N_{k+1} of agents around t_k. If we are able to do so, then we have computed the left optimal aggregate gap-minimizing assignment, as the gap is additive. In case N_{k+1} cannot be accommodated exactly around t_k, then this means that N_{k+1} is placed adjacent to the rightmost agent in N_k. We then move N_{k+1} one slot leftwards with as many agents moved one slot to the left as to accommodate N_{k+1}. Note that, as the assignment for $N_1, \ldots, N_k$ was left optimal, this move only increases the aggregate gap for agents in $N_1, \ldots, N_k$ but decreases the aggregate gap for agents in N_{k+1}. If such a move does not increase the aggregate gap of agents $N_1, \ldots, N_{k+1}$, we allow for such a move. We repeat these moves, until N_{k+1} cannot be shifted leftwards without increasing the aggregate gap of all the agents in $N_1, \ldots, N_{k+1}$. Thus, if the assignment in iteration $k+1$ is aggregate gap minimizing then it is left-respecting. It remains to be shown that the assignment in iteration $k+1$ is aggregate gap-minimizing. Let j be the smallest number such that N_j are placed contiguously to N_{k+1}. Observe that if the assignment in iteration $k + 1$ is not aggregate gap minimizing then there exists another aggregate gap minimizing (and left-respecting) assignment for which the block $N_j \cup \cdots \cup N_{k+1}$ is separated by at least one empty slot. However, the assignment to the left of such an empty slot is identical to the one in iteration k, contradicting that the algorithm produced the outcome in iteration $k + 1$. ■

The next result states the speed of the above algorithm.

Theorem 2. *The Neat Shifting Algorithm runs in $O(n^3)$ time.*

Proof. In each iteration of the outer while loop, one type is handled. There can be at most $O(n)$ iterations of this while loop. Then agents may need to be moved leftwards in the second while loop. Agents within the new type N_{k+1}, can be at most $O(n)$ away from their target because of the agents already allocated to slots. The type N_{k+1} has to move at most $O(n)$ to the left, as well as other agents that need to yield room for class N_{k+1}. Therefore, the second while loop runs at most $O(n)$ times. After each leftward shift, we check whether the new gap is more than previous gap which takes $O(n)$. Hence, the overall algorithm takes $O(n^3)$ time. ■

A standard combinatorial optimization approach would allow to compute an aggregate gap-minimizing assignment by a reduction to a maximum weight matching in which each vertex (i, s) represents an agent-slot pair and each edge is $|t_i - s|$. Such an algorithm takes $O(n^3)$ time, only if the number of slots is $O(n)$. The overall running time can be much worse as we do not impose any assumption on the number of slots.

The Neat Shifting Algorithm presented above thus yields a tailor-made algorithm that is faster than general algorithms for our (line) setting where there can be many more available slots than the number of agents.

[5] When we consider neat assignments, we are only concerned with the positions of the agent types and do not distinguish between assignments where agents within the same type swap positions.

5. Probabilistic aggregate gap-minimizing assignments

Deterministic assignment is usually criticized on equity grounds. The alternative is to deal with probabilistic assignment, which arises when randomizing over deterministic allocations. Formally, a *probabilistic assignment* is a specification of marginal probabilities for each agent over slots. For a given problem, a *lottery* is a probability distribution over deterministic allocations. We endorse the approach in the seminal contribution of Bogomolnaia and Moulin (2001) and assume that each agent cares only about his marginal distribution over outcomes (gaps, in our framework). A probabilistic assignment is (aggregate) gap-minimizing if there exists a lottery, target-ordered with it, such that each allocation in its support is (aggregate) gap-minimizing.

A *solution* is a mapping from problems to probabilistic assignments. We say that a solution satisfies a property if each probabilistic assignment it gives rise to satisfies that property. We impose from the outset that solutions satisfy the following three basic fairness requirements: *Equal treatment of equals*, which says that agents with the same target should get the same probability distribution over slots; *Slot invariance*, meaning that the solution does not depend on the labeling of slots; and *Symmetry*, which says that if we "flip" the problem from left to right, the corresponding solution flips too.

The so-called *random priority* (RP), which assigns objects based on a random ordering of the agents is a focal solution. It is, however, in conflict with efficiency (and, in the case of our setting, even with weak forms of it).

Hougaard et al. (2014) propose what they dubbed *Modified RP*, which is not only sd-efficient but also aggregate gap-minimizing. The modification behaves as RP by ordering agents randomly and then assigning them to slots sequentially, but with the important modification of doing so in a way that will ensure that the allocation following each iteration in the algorithm will be aggregate gap-minimizing, among those agents already assigned.

For the sake of completeness we present the definition of Modified RP as presented by Hougaard et al. (2014).

Modified RP: Order agents randomly. Assign the first agent his target. Assign the second agent a most preferred free slot; if there are two such slots pick one of them with equal probability, ... Suppose that $k-1$ agents have been assigned, and consider now the next agent (k) in the order. If his target is free, assign him the target and go to the next agent. If his target is not free, we consider two allocations constructed as follows:

(a) Assign k to the slot furthest to the right that is occupied by some agent i with target to the left of k and who is assigned to an outcome to the right of his own target (if no such outcome exists, assign k to the first free slot to the left of his target, and go to (b)). Assign agent i to the slot furthest to the right that is occupied by some agent j with target to the left of i, and who is assigned to an outcome to the right of his own target (if no such outcome exists, assign j to the first free slot to the left of his target, and go to (b)), etc. until some agent has been assigned the first free slot to the left.

(b) Make the symmetric counterpart construction involving agents having target to the right of k's target, which stops when an agent has been assigned the first free slot to the right of k's target.

As noted in Hougaard et al. (2014), it is clear from the construction of the modified RP algorithm that the solution it induces satisfies the three basic fairness requirements imposed from the outset: equal treatment of equals, slot invariance, and symmetry.

The next result shows Modified RP's computational speed.

Theorem 3. *Modified RP runs in $O(n^3)$ time.*

Proof. There are n iterations in which a new agent is allocated a slot and previous agents may be reallocated. The new agent k has to scan at most $O(n)$ slots from his target to either find an empty slot or replace an agent. Each replaced agent then has to scan slots $O(n)$ to the left of his current position to find an empty slot or replace an agent. In the worst case, there can be $O(n)$ replacements. Hence, the overall running time is $O(n^3)$. ■

The statement above shows that Modified RP is faster than using general maximum weight matching algorithms even when the number of slots is $O(n)$. Modified RP and the Neat Shifting algorithm have the same running time. However, the advantage of the Neat Shifting algorithm is that it can be easily modified to return gap-egalitarian assignments. We will present this idea in the next section.

Similar to Modified RP, we can adapt the Neat Shifting algorithm to obtain a randomized algorithm that satisfies symmetry. We start with a deterministic aggregate gap-minimizing assignment. With probability 1/2, we get such an assignment exactly as described in the mentioned algorithm. In order to ensure flip invariance, we also consider, with probability 1/2, the assignment derived from the algorithm replacing its convention, to consider right instead of left.[6] Once we have the deterministic (initial) assignment, we reassign slots uniformly at random among clones in each N_i. The next statement follows directly from Theorem 2.

Theorem 4. *The Probabilistic Neat Shifting Algorithm runs in $O(n^3)$ time.*

There is another natural way to extend RP to our setting so that the extension is also aggregate gap-minimizing. More precisely, take a permutation π of agents uniformly at random. For each $\pi(i)$, check (using a maximum weight perfect matching algorithm) whether $\pi(i)$ can be given a most preferred slot while ensuring that there exists an aggregate gap-minimizing assignment in which each agent before $\pi(i)$ in the permutation gets his utility level. If not, then $\pi(i)$ tries for the next most preferred slots until there exists an aggregate gap-minimizing assignment in which each agent before $\pi(i)$ in the permutation gets his utility level. The process is repeated until all agents in the permutation are dealt with. Then, an allocation can be computed by suitably computing a maximum weight perfect matching. Such an algorithm requires using a reduction to a maximum weight matching. The algorithm takes $O(n^5)$ time even if the number of slots is $O(n)$. Note however that the extension of RP described above is not restricted to the line assignment setting. It yields a (probabilistic) welfare maximizing solution, which captures RP in the sense that subsequent agents cannot change the utility level of a previous agent.

6. Egalitarian assignments

We now focus on *egalitarian* rather than aggregate gap-minimizing assignments. More precisely, we have concentrated so far on making the aggregate gap of an assignment as small as possible. Now, we concentrate on assignments whose maximum gaps are as small as possible (and, among them, on assignments whose second largest gaps are as small as possible, etc.).

As shown with Example 1, and the allocations x and y depicted afterwards, a gap-egalitarian assignment may not be (aggregate) gap-minimizing and vice versa. More precisely, for the problem

[6] More precisely, replace "left" by "right" in the sentence "In the solution computed all agents in N_i will be on the left of all agents in N_j for $i < j$" at the beginning of the description of the algorithm.

in which 5 agents wish to be located at slot 5, whereas 2 agents wish to be located at slot 7, consider two allocations. In allocation y, two of the first 5 agents and one of the 2 last agents are assigned to a slot at a distance 2 of their respective targets, whereas two of the remaining first 5 agents, and the remaining agent of the last 2 are assigned to a slot at a distance 1 of their targets. This implies that the maximum gap is 2 and, as a result, y is a gap-egalitarian assignment. In allocation x, the entire contiguous segment is moved one slot to the left, which makes the aggregate gap reduce from 9 to 8 (but at the cost of making one agent face a gap of 3). Thus, gap egalitarianism does not imply aggregate gap minimization and, conversely, aggregate gap minimization does not imply gap egalitarianism.[7]

We then explore features of gap-egalitarian assignments. Previously, we showed that there always exists an aggregate gap-minimizing assignment that is neat. We show now that neatness is a necessary condition for gap-egalitarian assignments.

Lemma 4. *Each gap-egalitarian assignment is neat.*

Proof. We first note that if a gap-egalitarian assignment is target-ordered then it is contiguous. Assume otherwise. Then, there exist agents of at least one type that have empty slots between them. Now, the target is either on the gap, to the left of the gap, or to right of the gap. In all three cases, the maximal gap and the lexicographic vector can be decreased by moving agents one slot towards the gap.

Next, suppose there exists a gap-egalitarian assignment that is not target-ordered, i.e., there exist two agents where i is located to the left of j but t_i is to the right of t_j. We claim that the lexicographic vector decreases when swapping the agents. Indeed, let x be the original assignment and x' be the assignment after i and j swap slots. If i and j are both assigned slots at or in between their target, they both benefit from a swap, contradicting that x is gap-egalitarian. If i and j are both assigned to a slot on the opposite side of the other agent's target, a swap will reduce the largest gap for these two agents, contradicting that x is gap-egalitarian. Similarly, if both agents are located on the right of j's target, or if both agents are located on the left of i's target, a swap will reduce the largest gap for these two agents, and we are done. It remains to consider the cases where one agent is located at, or in between, the two targets, and the other agent is located outside this area. Without loss of generality, let i be located to the right of j's target and j be located at, or in between, i and j's targets. Then, switching slots, i would reduce the gap (i.e., $g_i(x') < g_i(x)$). Furthermore, j may increase his gap but it would still be smaller than i's original gap (i.e., $g_j(x') < g_i(x)$). Thus, a contradiction with the fact that x is gap-egalitarian. The remaining case is symmetric. We conclude that every gap-egalitarian assignment must be neat. ■

Lemma 5. *There exists a unique (subject to intra-type swapping) left-respecting gap-egalitarian assignment.*

Proof. For a given problem, let $\mathcal{A}$ denote the set of its neat gap-egalitarian assignments. For each assignment x, let $\ell(N_k, x)$ be the leftmost slot occupied by an agent in type N_k. We claim that there exists $y \in \mathcal{A}$ such that, for each N_k, $\ell(N_k, y) = \min_{x\in\mathcal{A}} \ell(N_k, x)$. Let $i = 1, \ldots, k'$ be an index on types where 1 is the type with target furthest to the left and k' is the type with target furthest to the right. We proceed by induction on types.

Type 1. Obviously, there exists an assignment $y \in \mathcal{A}$ where $\ell(N_1, y) = \min_{x\in\mathcal{A}} \ell(N_1, x)$.

Induction hypothesis. Assume there exists an assignment $y \in \mathcal{A}$ where $\ell(N_k, y) = \min_{x\in\mathcal{A}} \ell(N_k, x)$, for $i = 1, \ldots, k$.

Type $k + 1$. Assume, by contradiction, that $\ell(N_{k+1}, y) \neq \min_{x\in\mathcal{A}} \ell(N_{k+1}, x)$. Then, there exists an assignment $z \in \mathcal{A}$ where $\ell(N_{k+1}, z) = \min_{x\in\mathcal{A}} \ell(N_{k+1}, x)$. Now, define the assignment w such that it is identical to y for types $1, \ldots, k$ and identical to z for types $k+1, \ldots, k'$. We claim that $w \in \mathcal{A}$. Assume, by contradiction, that this is not the case. Then, going from w to y would give a more egalitarian outcome, hence contradicting that $z \in \mathcal{A}$. ■

We show in the following result that the Neat Shifting Algorithm, with the target goal of gap-egalitarianism, returns a gap-egalitarian assignment.

Theorem 5. *The Neat Shifting Algorithm with goal gap-egalitarian yields a gap-egalitarian assignment.*

Proof. By Lemma 5, there exists a unique left-respecting gap-egalitarian assignment. We need to prove that this is the assignment returned by the algorithm. Note that the assignment produced by the algorithm is neat and each agent class is at most n slots away from its target. If an agent's class is not placed exactly around its target, it is because the class is forced to shift and make space to accommodate a latter class of agents with a target that is near enough. The only reason the assignment may not be gap-egalitarian is if the contiguous blocks of agents are not placed optimally. We prove by induction that assignments produced by the Neat Shifting Algorithm are left-respecting and gap-egalitarian.

First note that, in iteration 1, the algorithm produces a left-respecting neat gap-egalitarian assignment. Suppose this is the case for iterations $1, \ldots, k$. Now, we aim to show that the outcome of iteration $k + 1$ is left-respecting and gap-egalitarian. Indeed, we first try to place the block of N_{k+1} of agents around t_k. If we are able to do so, then we have computed the left-respecting gap-egalitarian assignment. If N_{k+1} cannot be accommodated exactly around t_k, then this means that N_{k+1} is placed adjacent to the right most agent in N_k. We then move N_{k+1} one slot leftwards, with as many agents moved one slot to the left as necessary to accommodate N_{k+1}. Note that since the assignment for $N_1, \ldots, N_k$ was left-respecting, this move only increases the gap-egalitarian goal for agents in $N_1, \ldots, N_k$, but decreases the gap-egalitarian goal for agents in N_{k+1}. If such a move decreases, or at least does not increase the gap-egalitarian goal of agents $N_1, \ldots, N_{k+1}$, we allow for such a move. We repeat these moves, until N_{k+1} cannot be shifted leftwards without increasing the gap-egalitarian goal of all the agents in $N_1, \ldots, N_{k+1}$. Thus, if the assignment in iteration $k + 1$ is gap-egalitarian then it is left-respecting. It remains to be shown that the assignment in iteration $k+1$ is gap-egalitarian. Let j be the smallest number such that N_j to N_{k+1} is placed contiguously. Observe that if the assignment in iteration $k + 1$ is not gap-egalitarian then there exists another gap-egalitarian (and left-respecting) assignment for which the block $N_j \cup \cdots \cup N_{k+1}$ is separated by at least one empty slot. However, the assignment to the left of such an empty slot is identical to the one in iteration k contradicting that the algorithm produced the outcome in iteration $k + 1$. ■

The only difference between the Neat Shifting Algorithm for aggregate gap minimization and gap egalitarianism is that, when shifting the latest agent type set leftwards, the ℓ and r values updates depend on whether the aggregate gap remains the same, or whether the lexicographic signature remains the same. Since the lexicographic signatures of two allocations can be compared in $n \log n$ time, it follows that the Neat Shifting Algorithm for egalitarian gap minimization takes time $O(n^3 \log n)$.

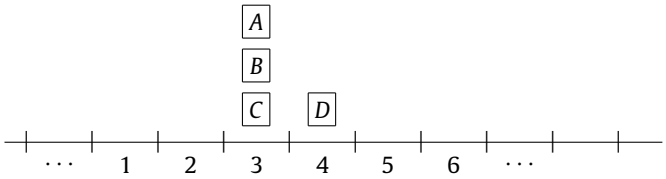
As with the aggregate gap minimizing algorithm, it is of course possible to start from the right instead of the left, and in this way an alternative gap-egalitarian allocation can be obtained. Given a deterministic gap-egalitarian allocation, it is possible to construct, from these two gap-egalitarian allocations, a

[7] Note that, in this example, one cannot find a solution that is both aggregate-gap minimizing and gap-egalitarian.

probabilistic assignment which satisfies equal treatment of equals, slot invariance, and symmetry, by an appropriate randomization over these two allocations and slots associated with each target. However, such a probabilistic assignment fails to be a gap-egalitarian allocation of the agents' expected gaps. Example 2 illustrates this.

Example 2. $[\{A, B, C\} : 3; \{D\} : 4]$.

Consider the problem $[\{A, B, C\} : 3; \{D\} : 4]$. In this case, both the left-oriented and right-oriented versions of the gap-egalitarian allocation algorithm will give agent D gap 1. However, the gap-egalitarian allocation of expected gaps will give each agent an expected gap of 3/4. This could be obtained for example in the following way: give D his target (i.e., slot 4) with probability 1/4 and otherwise give him slot 5. When he gets his target slot, the other agents randomize over slots 2, 3, and 5. When agent D gets slot 5, the remaining agents randomize over slots 2, 3, and 4. The expected gap of A, B, and C is $(\frac{3}{4})(\frac{1}{3})(2) + (\frac{3}{4})(\frac{1}{3})(2) = 3/4$. Note also that the gap-egalitarian (probabilistic) allocation does not imply neatness (or, more precisely, it does not need to be target ordered).



A *gap-egalitarian allocation of expected gaps* can be computed by setting up a series of linear programs. In the first linear program, a fractional (probabilistic assignment) is computed in which the maximum expected gap is minimized. In the subsequent linear programs, the next worst expected gap is minimized while maintaining the tight constraints from the previous linear programs (Garg et al., 2010).

7. Conclusion

In this paper, we have focused on the problem of assigning agents to a line, originally introduced by Hougaard et al. (2014). We have introduced a new algorithm to determine aggregate gap-minimizing assignments for such a problem and have studied its computational speed. The algorithm represents an advance with respect to the original algorithm by Hougaard et al. (2014), which determined whether an assignment was aggregate gap-minimizing or not.

We have also presented a counterpart algorithm to the previous one to determine gap-egalitarian (rather than aggregate gap-minimizing) assignments for such a problem.

Our analysis has also been extended from deterministic assignments to probabilistic assignment. In the latter scenario, we have considered new methods, as well as existing modifications of the focal random priority method to guarantee gap minimization.

References

Abdulkadiroğlu, A., Sönmez, T., 2003. Ordinal efficiency and dominated sets of assignments. J. Econom. Theory 112 (1), 157–172.
Aziz, H., Biro, P., Lang, J., Lesca, J., Monnot, J., 2016. Optimal reallocation under additive and ordinal preferences. In: Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS, pp. 402–410.
Aziz, H., Brandt, F., Harrenstein, P., 2013. Pareto optimality in coalition formation. Games Econom. Behav. 82, 562–581.
Bogomolnaia, A., Moulin, H., 2001. A new solution to the random assignment problem. J. Econom. Theory 100 (2), 295–328.
Bouveret, S., Chevaleyre, Y., Maudet, N., 2016. Fair allocation of indivisible goods. In: Brandt, F., Conitzer, V., Endriss, U., Lang, J., Procaccia, A.D. (Eds.), Handbook of Computational Social Choice. Cambridge University Press, (Chapter 12).
Burkhard, R., Dell'Amico, M., Martello, S., 2009. Assignment Problems. SIAM.
Chun, Y., Park, B., A graph theoretic approach to the slot allocation problem. Soc. Choice Welf. (forthcoming) http://dx.doi.org/10.1007/s00355-016-0975-y.
Damamme, A., Beynier, A., Chevaleyre, Y., Maudet, N., 2015. The power of swap deals in distributed resource allocation. In: Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent Systems. (AAMAS). IFAAMAS, pp. 625–633.
Demko, S., Hill, T.P., 1988. Equitable distribution of indivisible objects. Math. Social Sci. 16, 145–158.
Garg, N., Kavitha, T., Kumar, A., Mehlhorn, K., Mestre, J., 2010. Assigning papers to referees. Algorithmica 58, 119–136.
Hofstee, W.K.B., 1990. Allocation by lot: a conceptual and empirical analysis. Soc. Sci. Inf. 29 (4), 745–763.
Hougaard, J.L., Moreno-Ternero, J.D., Østerdal, L.P., 2014. Assigning agents to a line. Games Econom. Behav. 87, 539–553.
Hylland, A., Zeckhauser, R., 1979. The efficient allocation of individuals to positions. J. Polit. Econ. 87 (2), 293–314.
Kasajima, Y., 2013. Probabilistic assignment of indivisible goods with single-peaked preferences. Soc. Choice Welf. 41 (1), 203–215.
Katta, A.-K., Sethuraman, J., 2006. A solution to the random assignment problem on the full preference domain. J. Econom. Theory 131 (1), 231–250.
Lang, J., Rothe, J., 2015. Fair division of indivisible goods. In: Economics and Computation an Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division. Springer, pp. 493–550.
Nguyen, N.-T., Nguyen, T.-T., Roos, M., Rothe, J., 2014. Computational complexity and approximability of social welfare optimization in multiagent resource allocation. Auton. Agents Multi-Agent Syst. 28 (2), 256–289.
Procaccia, A.D., Tennenholtz, M., 2013. Approximate mechanism design without money. ACM Trans. Econ. Comput. 1 (4).
Zhou, L., 1990. On a conjecture by Gale about one-sided matching problems. J. Econom. Theory 52 (1), 123–135.